

5

# Enumerados y arrays

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Raquel Hervás Ballesteros  
Virginia Francisco Gilmartín  
Facultad de Informática  
Universidad Complutense



# Tipos

---

## ✓ Tipos simples

- ❖ Estándar: `int`, `float`, `double`, `char`, `bool`  
Conjunto de valores predeterminado
- ❖ Definidos por el usuario: *enumerados*  
Conjunto de valores definido por el programador

## ✓ Tipos estructurados

- ❖ Colecciones homogéneas: *arrays*  
Todos los elementos de la colección son de un mismo tipo
- ❖ Colecciones heterogéneas: *estructuras*  
Elementos de la colección de tipos distintos



# Tipos declarados por el usuario

Definición de tipo por el usuario:

```
typedef descripción nombre_de_tipo;
```



*Identificador válido*

```
typedef descripción tMiTipo;  
typedef descripción tMoneda;  
typedef descripción tTiposDeCalificacion;
```



Nombres de tipos propios:

t minúscula seguida de una o varias palabras capitalizadas

Los colorearemos en naranja, para remarcar que son tipos



# Tipos enumerados. Ejemplo

¿Cómo representarías las notas (NP, SS, AP, NT, SB, MH) de una asignatura?

Entero:

```
const int NP = 0;  
const int SS = 1;  
const int AP = 2;  
const int NT = 3;  
const int SB = 4;  
const int MH = 5;
```



```
int nota;
```

String:

```
const string NP = "No presentado";  
const string SS = "Suspendido";  
const string AP = "Aprobado";  
const string NT = "Notable";  
const string SB = "Sobresaliente";  
const string MH = "Matricula";
```



```
string nota;
```

Enumerados:

```
typedef enum { np, ss, ap, nt, sb, mh } tColor;  
tColor nota;
```



# Declaración de variables de tipo enumerado

```
typedef enum { centimo, dos_centimos, cinco_centimos,  
              diez_centimos, veinte_centimos,  
              medio_euro, euro } tMoneda;
```

En el ámbito de la declaración, podremos declarar variables de tipo **tMoneda**

```
tMoneda moneda1, moneda2;
```

Cada variable de ese tipo contendrá alguno de los símbolos que aparecen en la descripción del tipo

```
moneda1 = dos_centimos;
```

```
moneda2 = euro;
```

(Internamente se usan enteros)

moneda1

dos\_centimos

moneda2

euro



# Tipos enumerados

Conjunto de valores ordenado (posición en la enumeración)

```
typedef enum { lunes, martes, miercoles, jueves,  
viernes, sabado, domingo } tDiaSemana;
```

```
tDiaSemana dia;
```

```
...
```

```
if (dia == jueves)...
```

```
bool noLaborable = (dia >= sabado);
```

```
lunes < martes < miercoles < jueves  
< viernes < sabado < domingo
```

No admiten operadores de incremento y decremento

Emulación con moldes:

```
int i = int(dia); // ¡dia no ha de valer domingo!  
i++;  
dia = tDiaSemana(i);
```



# Entrada/salida para tipos enumerados (I)

```
typedef enum { enero, febrero, marzo, abril, mayo,  
             junio, julio, agosto, septiembre, octubre,  
             noviembre, diciembre } tMes;
```

```
tMes mes;
```

✓ Lectura de la variable mes:

```
cin >> mes;
```

- Se espera un valor entero
- No se puede escribir directamente **enero** o **junio**

✓ Y si se escribe la variable en la pantalla:

```
cout << mes;
```

- Se verá un número entero

→ Código de entrada/salida específico



# Entrada/salida para tipos enumerados (II)

---

Los enumerados no son más que un conjunto de constantes que hacen más comprensibles los programas

Al igual que los nombres de constantes pueden utilizarse en los programas pero no pueden ser leídos ni escritos

Los valores de los enumerados (**enero**, **febrero**, **céntimo**...) son “etiquetas” que sustituyen a enteros (como las constantes de tipo entero), internamente para el compilador son sólo enteros en el rango de valores definido por el enumerado





# Lectura del valor de un tipo enumerado

```
typedef enum { enero, febrero, marzo, abril, mayo, junio, julio,  
             agosto, septiembre, octubre, noviembre, diciembre } tMes;
```

```
int op;  
cout << " 1 - Enero"      << endl;  
cout << " 2 - Febrero"    << endl;  
cout << " 3 - Marzo"      << endl;  
cout << " 4 - Abril"      << endl;  
cout << " 5 - Mayo"       << endl;  
cout << " 6 - Junio"      << endl;  
cout << " 7 - Julio"      << endl;  
cout << " 8 - Agosto"     << endl;  
cout << " 9 - Septiembre" << endl;  
cout << "10 - Octubre"    << endl;  
cout << "11 - Noviembre"  << endl;  
cout << "12 - Diciembre"  << endl;  
cout << "Numero de mes: ";  
cin >> op;  
tMes mes = tMes(op - 1);
```



# Escritura de variables de tipos enumerados

```
typedef enum { enero, febrero, marzo, abril, mayo, junio, julio,  
             agosto, septiembre, octubre, noviembre, diciembre } tMes;
```

```
if (mes == enero) {  
    cout << "enero";  
}  
else if (mes == febrero) {  
    cout << "febrero";  
}  
else if (mes == marzo) {  
    cout << "marzo";  
}  
...  
else if (mes == diciembre) {  
    cout << "diciembre";  
}
```

También podemos utilizar una instrucción switch



# Ejemplo de tipos enumerados (I)

fechas.cpp

```
#include <iostream>
using namespace std;
```



Si los tipos se usan en varias funciones, los declaramos antes de los prototipos

```
typedef enum { enero, febrero, marzo, abril, mayo,
             junio, julio, agosto, septiembre, octubre,
             noviembre, diciembre } tMes;
```

```
typedef enum { lunes, martes, miercoles, jueves,
             viernes, sabado, domingo } tDiaSemana;
```

```
string cadMes(tMes mes);
string cadDia(tDiaSemana dia);
```

```
int main() {
    tDiaSemana hoy = lunes;
    int dia = 21;
    tMes mes = octubre;
    int anio = 2013;
    ...
}
```



# Ejemplo de tipos enumerados (II)

```
// Mostramos la fecha
cout << "Hoy es: " << cadDia(hoy) << " " << dia
      << " de " << cadMes(mes) << " de " << anio
      << endl;

cout << "Pasada la medianoche..." << endl;
dia++;
int i = int(hoy);
i++;
hoy = tDiaSemana(i);

// Mostramos la fecha
cout << "Hoy es: " << cadDia(hoy) << " " << dia
      << " de " << cadMes(mes) << " de " << anio
      << endl;

return 0;
}
```



# Ejemplo de tipos enumerados (III)

```
string cadMes(tMes mes) {  
    string cad;  
  
    if (mes == enero) {  
        cad = "enero";  
    }  
    else if (mes == febrero) {  
        cad = "febrero";  
    }  
    ...  
    else if (mes == diciembre) {  
        cad = "diciembre";  
    }  
  
    return cad;  
}
```

```
string cadDia(tDiaSemana dia);  
string cad;  
  
if (dia == lunes) {  
    cad = "lunes";  
}  
else if (dia == martes) {  
    cad = "martes";  
}  
...  
else if (dia == domingo) {  
    cad = "domingo";  
}  
  
return cad;  
}
```



# Ejercicio 1 Hoja de ejercicios Tema

*Declara un tipo enumerado `tCalificacion` con los valores `noPresentado`, `suspense`, `aprobado`, `notable`, `sobresaliente` y `matriculaDeHonor`. Luego, declara dos variables `nota1` y `nota2`, y guarda en ellas dos calificaciones numéricas de 0 a 10 (con un decimal) introducidas por el usuario. Asigna a dos variables `calif1` y `calif2`, de tipo `tCalificacion`, el valor que les corresponda, de acuerdo con los valores de las correspondientes variables numéricas que se han guardado en `nota1` y `nota2` (0: `noPresentado`). Finalmente, muestra cada nota numérica seguida de la calificación textual que le corresponde. Sólo mostrará las calificaciones si ambas notas están entre 0 y 10. El programa usará las funciones adecuadas.*

```
Nota 1: 0
Nota 2: 10
Nota 1: 0 (No presentado)
Nota 2: 10 (Matricula de Honor)
```



# Arrays. Definición

Un array es un tipo de dato para guardar un conjunto de elementos del mismo tipo (colección homogénea):

- ✓ Ventas de cada día de la semana
- ✓ Notas de los estudiantes de una clase
- ✓ Temperaturas de cada día del mes
- ✓ ...

Un array es un conjunto de variables de una misma clase. En lugar de declarar  $N$  variables

vLun	vMar	vMie	vJue	vVie	vSab	vDom
125.40	76.95	328.80	254.62	435.00	164.29	0.00

declaramos un array de  $N$  valores:

ventas	125.40	76.95	328.80	254.62	435.00	164.29	0.00
Índices →	0	1	2	3	4	5	6



# Arrays. Estructura y acceso

Los arrays tienen una estructura secuencial, cada elemento se encuentra en una posición (índice):

- ✓ Los índices son enteros positivos
- ✓ El índice del primer elemento siempre es 0
- ✓ Los índices se incrementan de uno en uno

ventas	125.40	76.95	328.80	254.62	435.00	164.29	0.00
	0	1	2	3	4	5	6

A cada elemento se accede a través de su índice:

`ventas[4]` accede al 5º elemento (contiene el valor 435.00)

```
cout << ventas[4];
```

```
ventas[4] = 442.75;
```



Datos de un mismo tipo base:  
Se usan como cualquier variable





# Declaración de tipos de arrays

```
typedef tipo_base nombre_tipo[tamaño];
```

Ejemplos:

```
typedef double tTemp[7];  
typedef short int tDiasMes[12];  
typedef char tVocales[5];  
typedef double tVentas[31];  
typedef tMoneda tCalderilla[15];
```



*Recuerda:* Adoptamos el convenio de comenzar los nombres de tipo con una t minúscula, seguida de una o varias palabras, cada una con su inicial en mayúscula



# Declaración de variables arrays

*tipo nombre;*

Ejemplos:

`tTemp` tempMax;

```
typedef double tTemp[7];
typedef short int tDiasMes[12];
typedef char tVocales[5];
typedef double tVentas[31];
```

tempMax

?	?	?	?	?	?	?
0	1	2	3	4	5	6

`tDiasMes` diasMes;

diasMes

?	?	?	?	?	?	?	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9	10	11

`tVocales` vocales;

vocales

?	?	?	?	?
0	1	2	3	4

`tVentas` ventasFeb;

ventasFeb

?	?	?	?	?	?	?	?	?	?	?	?	?	...	?
0	1	2	3	4	5	6	7	8	9	10	11	12		30



NO se inicializan los elementos automáticamente



# Capacidad de los arrays

---

- ✓ La capacidad de un array no puede ser alterada en la ejecución
- ✓ El tamaño de un array es una decisión de diseño:
  - ✓ En ocasiones será fácil (días de la semana)
  - ✓ Cuando pueda variar ha de estimarse un tamaño
    - Ni corto ni con mucho desperdicio (posiciones sin usar)
- ✓ STL (*Standard Template Library*) de C++: Colecciones más eficientes cuyo tamaño puede variar



# Acceso a los elementos de un array (I)

Cada elemento se accede a través de su índice (posición en el array):

```
nombre[índice]
typedef char tVocales[5];
tVocales vocales;
```

vocales	'a'	'e'	'i'	'o'	'u'
	0	1	2	3	4

✓ 5 elementos, índices de 0 a 4:

vocales[0]    vocales[1]    vocales[2]    vocales[3]    vocales[4]

✓ Procesamiento de cada elemento:

Como cualquier otra variable del tipo base

```
cout << vocales[4];
```

```
vocales[3] = 'o';
```

```
if (vocales[i] == 'e') ...
```



# Acceso a los elementos de un array (II)

¡IMPORTANTE!

¡No se comprueba si el índice es correcto!

*¡Es responsabilidad del programador!*

```
const int DIM = 100;  
typedef double tVentas[DIM];  
tVentas ventas;
```

Índices válidos: enteros entre 0 y DIM-1

ventas[0] ventas[1] ventas[2] ... ventas[98] ventas[99]

¿Qué es ventas[100]? ¿O ventas[-1]? ¿O ventas[132]?

¡Memoria de alguna otra variable del programa!



Define los tamaños de los arrays con constantes



# Secuencias en arrays

---

- ✓ Arrays completos: La secuencia ocupa todas las posiciones del array:
  - Tamaño array = longitud secuencia almacenada en el array
    - N elementos en un array de N posiciones
  - Recorremos el array desde la primera posición hasta la última
- ✓ Arrays no completos: La secuencia deja posiciones libres al final del array:
  - Tamaño array > longitud secuencia almacenada en el array
  - ¿Cómo identificar donde termina la secuencia?
    - Con centinela: Recorremos el array hasta encontrar el valor centinela



# Recorrido de arrays completos

- Todas las posiciones del array ocupadas

```
const int N = 10;  
typedef double tVentas[N];  
tVentas ventas;
```

...

ventas	125.40	76.95	328.80	254.62	435.00	164.29	316.05	219.99	93.45	756.62
	0	1	2	3	4	5	6	7	8	9

```
double elemento;  
// Inicialización  
// ¿Final?  
for (int i = 0; i < N; i++) {  
    // Obtener el elemento ...  
    elemento = ventas[i];  
    // Procesar el elemento  
    ...  
}
```



# Recorrido de arrays completos. Ejemplo (I)

---

Ejemplo: Media de un array de temperaturas

```
const int DIAS = 7;
typedef double tTemp[DIAS];
tTemp temp;
double media, total = 0;
...
for (int i = 0; i < DIAS; i++) {
    total = total + temp[i];
}
media = total / DIAS;
```

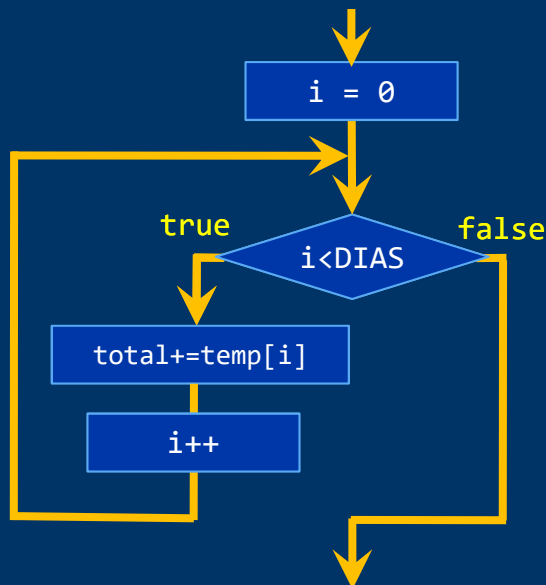




# Recorrido de arrays completos. Ejemplo (II)

12.40	10.96	8.43	11.65	13.70	13.41	14.07
0	1	2	3	4	5	6

```
tTemp temp;  
double media, total = 0;  
...  
for (int i = 0; i < DIAS; i++) {  
    total = total + temp[i];  
}
```



	Memoria
DIAS	7
temp[0]	12.40
temp[1]	10.96
temp[2]	8.43
temp[3]	11.65
temp[4]	13.70
temp[5]	13.41
temp[6]	14.07
media	?
total	84.62
i	7

...



# Recorrido de arrays completos. Ejemplo (III)

```
#include <iostream>
using namespace std;
```

mediatemp.cpp

```
const int DIAS = 7;
typedef double tTemp[DIAS];
```

```
double media(const tTemp temp);
```

```
int main() {
    tTemp temp;
    for (int i = 0; i < DIAS; i++) { // Recorrido del array
        cout << "Temperatura del día " << i + 1 << ": ";
        cin >> temp[i];
    }
    cout << "Temperatura media: " << media(temp) << endl;
    return 0;
}
...
```

Los usuarios usan de 1 a 7 para numerar los días  
La interfaz debe aproximarse a los usuarios,  
aunque internamente se usen los índices de 0 a 6



# Recorrido de arrays completos. Ejemplo (IV)

```
double media(const tTemp temp) {  
    double med, total = 0;  
  
    for (int i = 0; i < DIAS; i++) { // Recorrido del array  
        total = total + temp[i];  
    }  
    med = total / DIAS;  
  
    return med;  
}
```



Los arrays se pasan a las funciones como constantes  
Las funciones no pueden devolver arrays



# Recorrido de arrays. Ejemplo Fibonacci

*Array con los N primeros números de Fibonacci*

```
const int N = 50;
typedef long long int tFibonacci[N]; // 50 números
tFibonacci fib;

fib[0] = 1;
fib[1] = 1;
//Recorro el array para inicializar la serie Fibonacci
for (int i = 2; i < N; i++)
    fib[i] = fib[i - 1] + fib[i - 2];

//Recorro el array para mostrar la serie Fibonacci por pantalla
for (int i = 0; i < N; i++)
    cout << fib[i] << "  ";
```

fibonacci.cpp



# Recorrido de arrays. Ejemplo Digits (I)

## *Cuenta de valores con k dígitos*

digitos.cpp

❖ Recorrer una lista de N enteros contabilizando cuántos son de 1 dígito, cuántos de 2 dígitos, etcétera (hasta 5 dígitos)

❖ 2 arrays: array con los números y array de contadores

```
const int NUM = 100;  
typedef int tNum[NUM]; // Exactamente 100 números  
tNum numeros;  
const int DIG = 5;  
typedef int tDig[DIG]; //i --> números de i+1 dígitos  
tDig numDig = { 0 };
```

numeros	123	2	46237	2345	236	11234	33	999	...	61
	0	1	2	3	4	5	6	7		99
numDig	0	0	0	0	0					
	0	1	2	3	4					



# Recorrido de arrays. Ejemplo Digits (II)

- ❖ Función que devuelve el número de dígitos de un entero:

```
int digitos(int dato) {  
    // Al menos tiene un dígito  
    int n_digitos = 1;  
    // Recorremos la secuencia de dígitos...  
    while (dato >= 10) {  
        dato = dato / 10;  
        n_digitos++;  
    }  
    return n_digitos;  
}
```



# Recorrido de arrays. Ejemplo Digos (III)

```
#include <iostream>
using namespace std;
#include <cstdlib> // srand() y rand()
#include <ctime> // time()

int digitos(int dato);

int main() {
    const int NUM = 100;
    typedef int tNum[NUM]; // Exactamente 100 números
    const int DIG = 5;
    typedef int tDig[DIG];
    tNum numeros;
    tDig numDig = { 0 }; // Inicializa todo el array a 0

    srand(time(NULL)); // Inicia la secuencia aleatoria
    ...
}
```



# Recorrido de arrays. Ejemplo Digitos (III)

```
// Creamos la secuencia aleatoriamente
for (int i = 0; i < NUM; i++)
    numeros[i] = rand(); // Entre 0 y 32766

// Recorremos la secuencia de enteros y obtenemos el número
// de dígitos de cada entero de la secuencia
for (int i = 0; i < NUM; i++)
    numDig[digitos(numeros[i]) - 1]++;

// Recorremos la secuencia de contadores y
// mostramos el valor de cada contador
for (int i = 0; i < DIG; i++)
    cout << "De " << i + 1 << " díg. = " << numDig[i] << endl;

return 0;
}

int digitos(int dato) {
    ...
}
```





## Ejercicio 2 Hoja de ejercicios Tema 6

*Implementa un programa que pida al usuario las calificaciones de los 10 estudiantes de una clase y los guarde en un array (declara el tipo adecuado; se admiten decimales).*

*Después, mostrará esas calificaciones (cada una en una línea), seguida de la media del curso y del número de aprobados y suspensos.*

*Se usará una función para calcular la nota media y otra para el número de suspensos.*

```
Introduce la nota del estudiante 1: 7.75
Introduce la nota del estudiante 2: 8.25
Introduce la nota del estudiante 3: 10
Introduce la nota del estudiante 4: 9
Introduce la nota del estudiante 5: 3.25
Introduce la nota del estudiante 6: 6.30
Introduce la nota del estudiante 7: 2.25
Introduce la nota del estudiante 8: 7
Introduce la nota del estudiante 9: 6.95
Introduce la nota del estudiante 10: 8.25
Nota del estudiante 1 : 7.75
Nota del estudiante 2 : 8.25
Nota del estudiante 3 : 10.00
Nota del estudiante 4 : 9.00
Nota del estudiante 5 : 3.25
Nota del estudiante 6 : 6.30
Nota del estudiante 7 : 2.25
Nota del estudiante 8 : 7.00
Nota del estudiante 9 : 6.95
Nota del estudiante 10: 8.25

La nota media es: 6.90

Hay 2 suspensos y 8 aprobados
```



# Ejercicio 3a Hoja de ejercicios Tema

---

*Dado el siguiente tipo tVector para representar secuencias de N enteros:*

```
const int N = 10;  
typedef int tVector[N];
```

- a. Escribe una función que calcule y devuelva la suma de los elementos que se encuentran en las posiciones pares del array.*



# Recorrido de arrays no completos - centinela

- No todas las posiciones del array están ocupadas

```
const int N = 10;  
typedef double tArray[N];  
tArray datos; // Datos positivos: centinela = -1
```

...

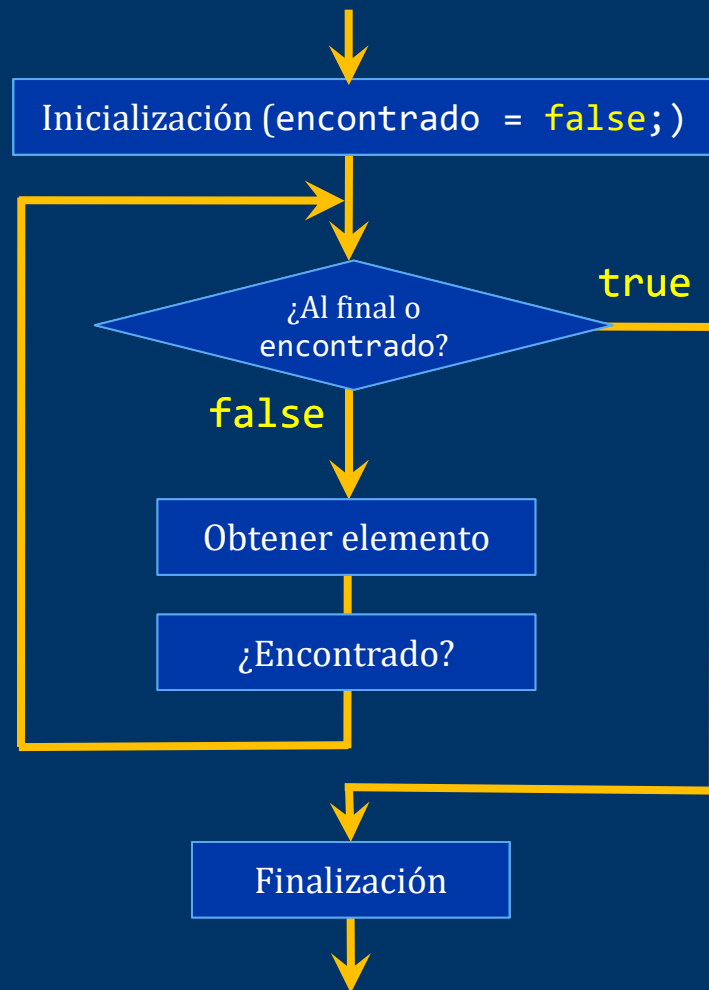
datos	125.40	76.95	328.80	254.62	435.00	164.29	316.05	-1.0		
	0	1	2	3	4	5	6	7	8	9

```
//inicialización  
int i = 0;  
// Obtener el elemento  
double elemento = datos[i];  
//¿final?  
while ((elemento != -1) && (i < N)) {  
    //procesar el elemento  
    cout << elemento;  
    i++;  
    elemento = datos[i];  
}
```



# Búsquedas en arrays

## Esquema de búsqueda



# Búsquedas en arrays completos

```
int buscado;  
cout << "Valor a buscar: ";  
cin >> buscado;
```

```
// Inicialización  
bool encontrado = false;  
int pos = 0;  
  
// ¿Final o encontrado?  
while ((pos < N) && !encontrado) {  
    // Obtener elemento //¿Encontrado?  
    if (lista[pos] == buscado)  
        encontrado = true;  
    else  
        pos++;  
}  
if (encontrado) // ...
```

```
const int N = 100;  
typedef int TArray[N];  
TArray lista;
```



# Búsqueda en arrays completos. Ejemplo

*¿Qué día las ventas superaron los 1.000 €?*

buscaarray.cpp

```
const int DIAS = 365; // Año no bisiesto
typedef double tVentas[DIAS];

int busca(const tVentas ventas) {
    // Índice del primer elemento mayor que 1000 (-1 si no hay)
    bool encontrado = false;
    int ind = 0;
    while ((ind < DIAS) && !encontrado) { // Esquema de búsqueda
        if (ventas[ind] > 1000) {
            encontrado = true;
        }
        else {
            ind++;
        }
    }
    if (!encontrado) {
        ind = -1;
    }
    return ind;
}
```



# Búsquedas en arrays incompletos - centinela

```
int buscado;
cout << "Valor a buscar: ";
cin >> buscado;
int pos = 0;
// Inicialización
bool encontrado = false;
// ¿Final o encontrado?
while ((array[pos] != centinela) && (pos < N) && !encontrado) {
    // Obtener elemento // ¿Encontrado?
    if (array[pos] == buscado)
        encontrado = true;
    else
        pos++;
}
if (encontrado) // ...
```

```
const int N = 10;
typedef int tArray[N];
tArray array;
const int centinela = -1;
```



# Arrays de tipos enumerados

```
const int Cuantas = 15;
typedef enum { centimo, dos_centimos, cinco_centimos,
             diez_centimos, veinte_centimos, medio_euro, euro } tMoneda;
typedef tMoneda tCalderilla[Cuantas];
string aCadena(tMoneda moneda);
// Devuelve la cadena correspondiente al valor de moneda

tCalderilla bolsillo; // Exactamente llevo Cuantas monedas
bolsillo[0] = euro;
bolsillo[1] = cinco_centimos;
bolsillo[2] = medio_euro;
bolsillo[3] = euro;
bolsillo[4] = centimo;
...
for (int moneda = 0; moneda < Cuantas; moneda++)
    cout << aCadena(bolsillo[moneda]) << endl;
```





# Copia de arrays

---

- ✓ No se pueden copiar dos arrays (del mismo tipo) con asignación:

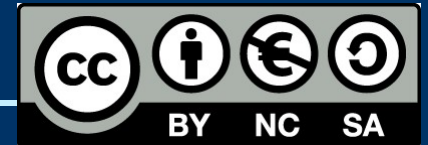
`array2 = array1;` *!!! NO COPIA LOS ELEMENTOS !!!*

- ✓ Han de copiarse los elementos uno a uno:

```
for (int i = 0; i < N; i++) {  
    array2[i] = array1[i];  
}
```






# Acerca de *Creative Commons*



## Licencia CC (*Creative Commons*)

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Material original elaborado por Luis Hernández Yáñez, con modificaciones de Raquel Hervás Ballesteros.

